DSWE Python Package
User Help Document

Prepared by Dr. Yu Ding's Research Group at
Texas A&M University

Last updated: July 29, 2022

Contact: yuding@tamu.edu

Table of Contents

1 Differences between DSWE's R package and Python package.

The functions in the two DSWE packages closely mirror each other, but with the following differences:

A. DSWE R package does not have the deep learning power curve function (DNNPowerCurve).
B. DSWE Python package does not have the spline-based power curve function (SplinePCFit).
C. DSWE Python package's BayesTreePowerCurve function uses BartPy, which is the Python implementation of BART. BartPy was not implemented by the original authors of BART and its results differ slightly from the BART function in R. So, the difference between the R and Python versions of the Bayes Tree power curve is larger than those between other power curve functions in the two packages.
D. For the time being, AMK in Python has to take the bandwidth parameters from the R package, as the optimal bandwidth selection algorithm, i.e., the direct plug-in (DPI) algorithm, is still being implemented. Once DPI is implemented, AMK in Python will be stand alone.
E. DSWE Python package does not have the Energy Decomposition function (deltaEnergy).

2 How do I install the package?

The package is available through PyPI, the official package repository of Python, as well as on GitHub. This package is tested on and is compatible with **Python of version 3.6 or above**. When using an earlier Python version, problems may arise.

The package should be installed using the standard `pip install dswe` command in command terminal:

```
pip install dswe
```

How to update the package to a new version?

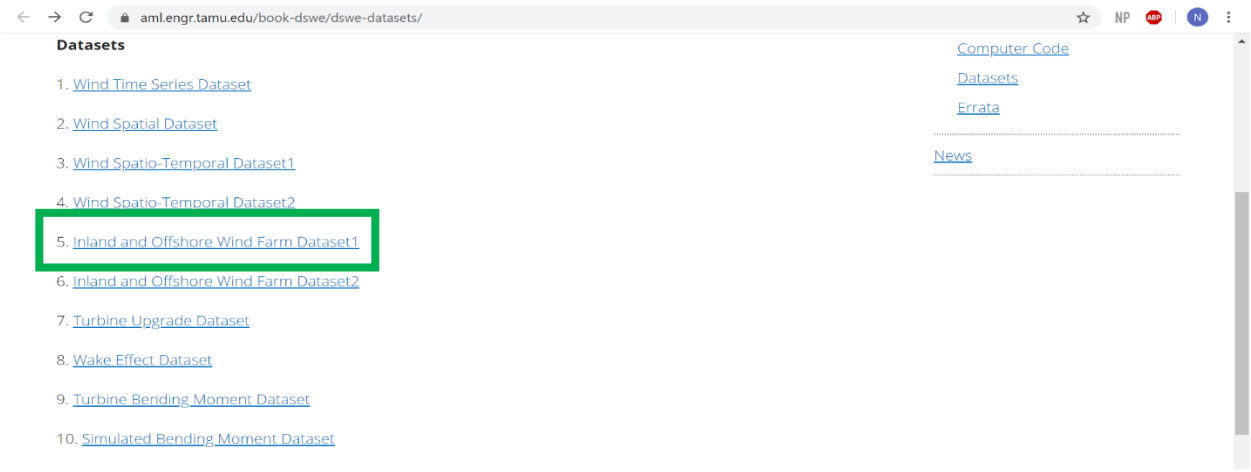
Use the following command in command:

```
pip install --upgrade dswe
```

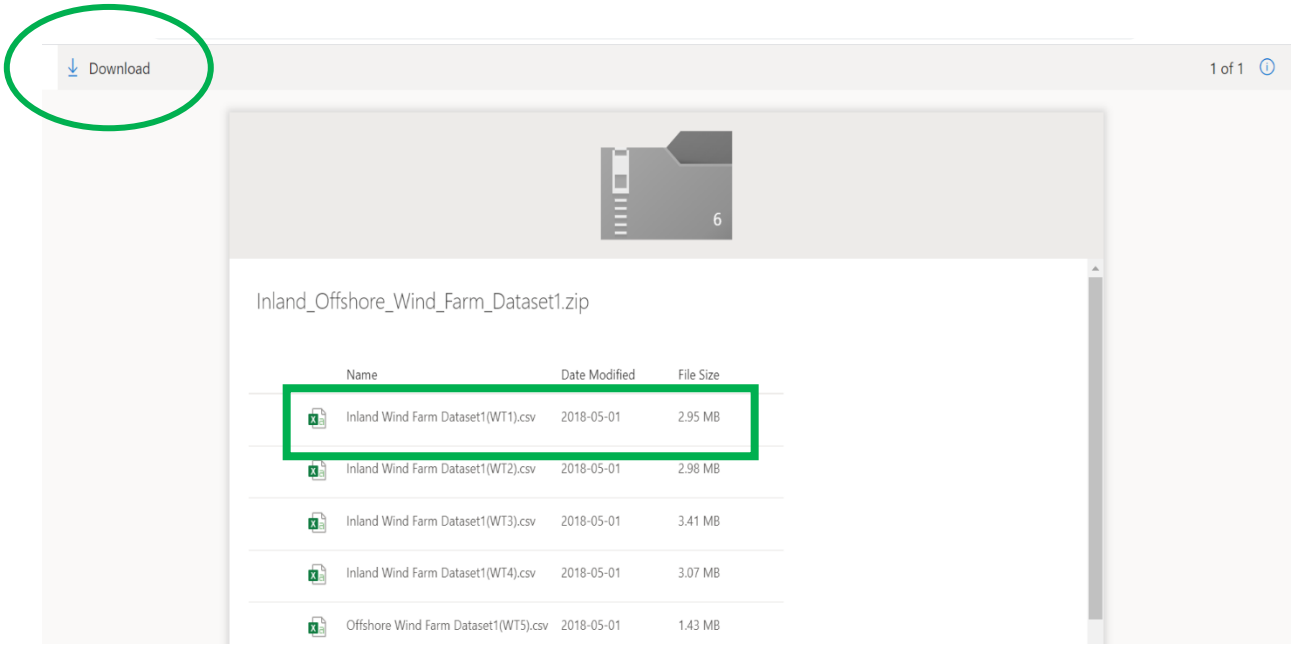
3 How to use KNN to fit a multi-dimensional power curve?

Step1: Download the sample data set as shown.

Visit site using the following link - *https://aml.engr.tamu.edu/book-dswe/dswe-datasets/*. The page looks like as shown below and select the option 5.



Download the sample data set as shown below in green boxes. After downloading, save the file in working directory.



Step 2: Set the path containing data set to a current working directory. Further load the package and import the data set as shown.

```
# loading the package
import pandas as pd
from dswe import KNNPowerCurve
```
[1] ✓ 1.2s

```
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
```
[2] ✓ 0.5s

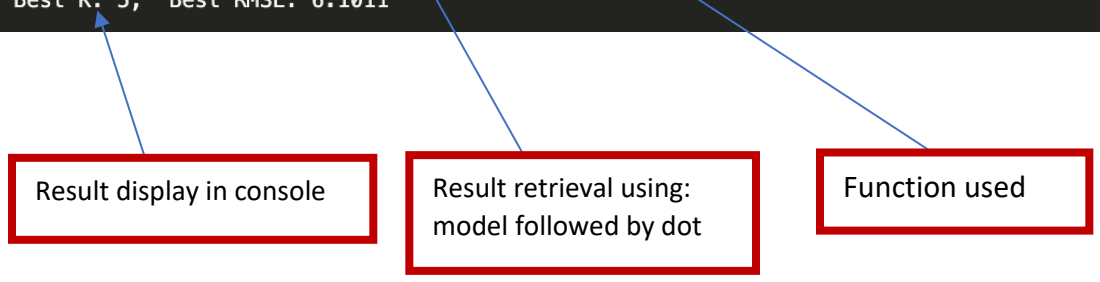
Step 3: Prepare the arguments to fit a power curve as shown below.

```
# feature selection
# selected first 1000 rows and column index 1,2,3,4,5 for training
X_train = df.iloc[:1000, 1:6]
y_train = df.iloc[:1000, 6]

# model fitting
knn_model = KNNPowerCurve()
knn_model.fit(X_train, y_train)

# result value retrieval
print(f"Best K: {knn_model.best_k},  Best RMSE: {knn_model.best_rmse}")
```
[3] ✓ 0.7s

```
Best K: 5,  Best RMSE: 6.1011
```



Note: - The RMSE reported is based on generalized cross validation on training set. To obtain a prediction on a test point, follow the next step.

Step 4: Prediction on a test point using the model generated from KNNPowerCurve and using the function predict as shown below.

How to select the best subset of variables in building a multi-dimensional power curve?

Step 1: The data set mentioned in previous questions will be used to demonstrate the usability of subset selections. The package is loaded and data is imported as shown.

```
# loading the package
import pandas as pd
from dswe import KNNPowerCurve
```
[1] ✓ 1.2s

```
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
```
[2] ✓ 0.5s

Step 2: Prepare the arguments to fit the KNN power curve and retrieve the best subset as shown.

```
# feature selection
# selected first 1000 rows and column index 1,2,3,4,5 for training
X_train = df.iloc[:1000, 1:6]
y_train = df.iloc[:1000, 6]

# model fitting
knn_model = KNNPowerCurve(subset_selection=True)
knn_model.fit(X_train, y_train)

# result value retrieval
print(f"Best subset: {knn_model.best_subset}")
```
[3] ✓ 0.2s

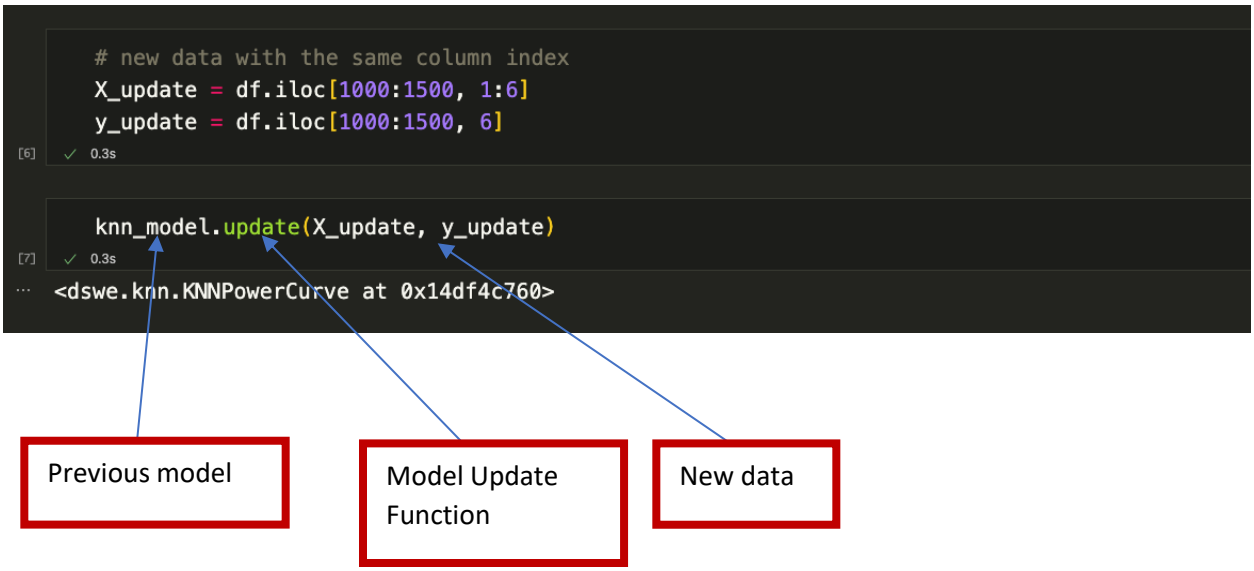
```
Best subset: [0, 1, 4, 2, 3]
```

Retrieve best subset column

KNN model fit with subset selection

Suppose one has built a KNN power curve model using some data. When the new data comes in, how can one update it periodically?

Step 1: Prepare the arguments to update the previously obtained model using new data as shown.



4 How to use temporal Gaussian process model to fit a multi-dimensional power curve?

Note:

The temporal Gaussian Process, or TempGP, is a Gaussian process-based model, and hence the inference (fitting) can be computationally expensive for a large data. In order to overcome this problem, we implemented an acceleration method for inference that utilizes a stochastic optimization technique known as Adam.

Adam has a few hyperparameters to tune. We tuned those parameters on many datasets to strike a balance between prediction accuracy and computation time. The fast computation is expected to be within two minutes on a single core of a modern computer compared to hours of computation required by original TempGP. The accelerated version is set as the default option.

We expect the fast computation results to be 5-10% worse than that of the original version of TempGP. If accuracy is extremely important to the user at the cost of orders of magnitude of increased computation time, then one can easily set the argument `fast_compuation` to `False` and `limit_memory` to None to use the original version of TempGP.

Example

For this example, we use data1 in the DSWE package.

Step 1: Load the dataset and create a training set with features and target variable. One can also create an array of time indices for the training data points. If the time indices are not created, the code assigns positive integers starting from 1 as the time indices. For example, if there are 100 training data points, the code will assign the time indices from 1 to 100.

```
[1] ✓ 1.6s
import numpy as np
import pandas as pd
from dswe import TempGP

[2] ✓ 0.5s
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')

[3] ✓ 0.2s
# feature selection, selected first 1000 rows and column index 1,2,3,4,5 for training
# index 0 has the time index
X_train = df.iloc[:1000, 1:6].values
y_train = df.iloc[:1000, 6].values
T_train = df.iloc[:1000, 0].values
```

Step 2: Call the TempGP function using the training data. There are two ways to call TempGP, with training time indices or without training time indices. As explained in Step #1, when

training time indices are not assigned, TempGP automatically assigns the time indices starting from 1.

```
    # model fitting
    tempGP_model = TempGP(opt_method='L-BFGS-B')
    tempGP_model.fit(X_train, y_train, T_train)
    tempGP_model.optim_result
[4]  ✓ 58.6s
...  {'estimated_params': {'theta': array([ 0.36927562, 44.72136687,  1.55740529,  0.08294084,  0.40005605]),
      'sigma_f': 24.11256789926995,
      'sigma_n': 1.2316988971478031,
      'beta': 26.75816024287659},
     'obj_val': None,
     'grad_val': None}
```

Step 3: Use the predict method to get predictions from the learned model. Again, one can either use just environmental input variables alone to predict the response, or also use the time indices of the test data points. In this example we show case for using just the environmental input variables and provide details of using time indices in the response to the next question (namely "How to update the training data in the TempGP model when new data is available?")

```
    # select 10 rows (different from train set)
    # and same column index as in the train set
    X_test = df.iloc[1000:1010, 1:6].values
    y_test = df.iloc[1000:1010, 6].values
    T_test = df.iloc[1000:1010, 0].values

    prediction = tempGP_model.predict(X_test, T_test)
    prediction
[6]  ✓ 0.1s
...  array([ 8.64410454,  0.16888329,  6.54695785,  4.29637959, -2.55994747,
            0.29491728,  1.54166811, 35.88894864, 23.20645634, 34.81744015])

    rmse = np.sqrt(np.square(prediction - y_test).mean())
    print ("RMSE: {}".format(rmse))
[7]  ✓ 0.3s
...  RMSE: 6.8149660546419675
```

How to update the training data in the TempGP model when new data is available?

Using the time indices for test data points improves the prediction accuracy of the TempGP model if the time indices of the test points are close to that of the training points, as the temporal dependence in response vanishes after a short period of time. Thus, we have provided another method called *update* to keep updating the training data as the new data becomes available.

```
## Predict both f(x) and g(t) using a rolling window data update with the help of
# update method of TempGP function.
pred_X_test = [0] * len(X_test) # list to store the rolling predictions
```

Step 1: Do an *i*-step ahead prediction given the input variables. In this example, we use the actual input variable values; if the input variable values are not available, replace with their forecast.

```
#starting a loop for doing the rolling predctions
for i in range(len(X_test)):
    # X_test[i], T_test[i] are input variables for time point i
    # replace with forecast when actual data not available.
    pred_X_test[i] = tempGP_model.predict([X_test[i]], [T_test[i]])
```

Step 2: Once the data for time point ' *i* ' is available, update the TempGP model using the *update* method of TempGP function. Use the fitted '*tempGP_model*' object on the training data points.

```
tempGP_model.update([X_test[i]], [y_test[i]], [T_test[i]], replace=True, update_model_F=False)
```

5 How to use AMK to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of AMK. The package is loaded and data is imported as shown.

```
# load the package
import numpy as np
import pandas as pd
from dswe import AMK
```
[1] ✓ 1.5s

```
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
```
[2] ✓ 0.4s

Step 2: Prepare the arguments to fit a power curve as shown.

```
# feature selection
X_train = df.iloc[:100, [1,2,3,4]].values
y_train = df.iloc[:100, 6].values
X_test = df.iloc[100:110, [1,2,3,4]].values

# parameters selection
bw = np.arange(1,5)*0.1  # provided the bandwidth corresponding to each column.
n_multi_cov = 3
fixed_cov = [0, 1]
cir_cov = [1]
```
[3] ✓ 0.4s

```
model = AMK(X_train, y_train, X_test, bw, n_multi_cov, fixed_cov, cir_cov)
print(np.round(model.predictions, 3))
```
[4] ✓ 0.2s

```
[ 9.141  5.119  3.177  3.475  7.766  9.95   6.104  3.185  3.234 10.986]
```

Predictions on the test data

6 How to use the Bayesian tree model to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of BayesTreePowerCurve. The package is loaded and data is imported as shown.

```
# load the package
import pandas as pd
import random
from dswe import BayesTreePowerCurve
```
[1] ✓ 1.1s

```
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
```
[2] ✓ 0.4s

Step 2: Prepare the arguments to fit a power curve as shown.

```
# feature selection and train-test set split
X_train = df.iloc[:100, [1,2,3,4]].values
y_train = df.iloc[:100, 6].values
X_test = df.iloc[100:110, [1,2,3,4]].values
y_test = df.iloc[100:110, 6].values
```
[3] ✓ 0.2s

```
random.seed(1)
clf = BayesTreePowerCurve()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
```
[ ]

```
pred
```
[5] ✓ 0.3s

```
array([ 5.07985548,  6.65056469,  3.46310427,  4.06533754,  4.12976387,
        3.62900425,  7.08586615,  9.97141396,  9.59230977, 10.96689502])
```

7 How to use the support vector machine to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of SVMPowerCurve. The package is loaded and data is imported as shown.

```
[1]  ✓ 1.2s
import pandas as pd
from dswe import SVMPowerCurve

[2]  ✓ 0.4s
# load the data set
df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
```

Step 2: Prepare the arguments to fit a power curve as shown.

```
[3]  ✓ 0.2s
# feature selection and train-test set split
X_train = df.iloc[:100, [1,2,3,4]].values
y_train = df.iloc[:100, 6].values
X_test = df.iloc[100:110, [1,2,3,4]].values
y_test = df.iloc[100:110, 6].values

[4]  ✓ 0.2s
clf = SVMPowerCurve()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
pred

...  array([ 3.82641404,  3.93851909,  6.44747753,  6.89594511,  8.66601587,
         9.33134373, 10.72174076, 19.14148183, 19.94223051, 20.11695447])
```

8 How to use the Deep Learning to fit a multi-dimensional power curve?

Step 1: The data set mentioned in previous question will be used to demonstrate the usability of DNNPowerCurve. The package is loaded and data is imported as shown.

```
[1]  import pandas as pd
     from dswe import DNNPowerCurve
     ✓ 1.2s

[2]  # load the data set
     df = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
     ✓ 0.5s
```

Step 2: Prepare the arguments to fit a power curve as shown.

```
[3]  X = df.iloc[:,1:6].values
     y = df.iloc[:,6].values
     X_train = X[:40000]
     y_train = y[:40000]
     X_test = X[40000:]
     y_test = y[40000:]
     ✓ 0.2s

[4]  dnn = DNNPowerCurve(train_all=True, save_fig=False)
     dnn.train(X_train, y_train)
     ✓ 49.1s
...  --Initiating training on the entire dataset--
     Everything done!!

[6]  ## Calculate RMSE
     print ("RMSE : {}".format(dnn.calculate_rmse(X_test, y_test)))
     ✓ 0.1s
...  RMSE : 7.264648889753912
```

9 How do the power curve functions compare with each other?

We tested the power curve functions implemented both in R and Python packages using Dataset#5 (Inland and Offshore Wind Farm Dataset1) of the *Data Science for Wind Energy* book. Please following the instruction of Step 1 to Question 3 "How to use KNN to fit a multi-dimensional power curve?" to download the datasets. There are six turbine datasets in Dataset#5, four from onshore turbines and two from offshore turbines.

The following power curve functions are tested: tempGP, AMK, KNN, BART, DNN, SSANONA, SVM, and the binning method (the IEC standard procedure).

The following table presents the root mean square error (RMSE) based on a randomized five-fold cross-validation. The power is normalized, with each turbine's rated power as 100%. So the values reported below are relative to the rated power. For instance, 0.1 means 10% of the rated power.

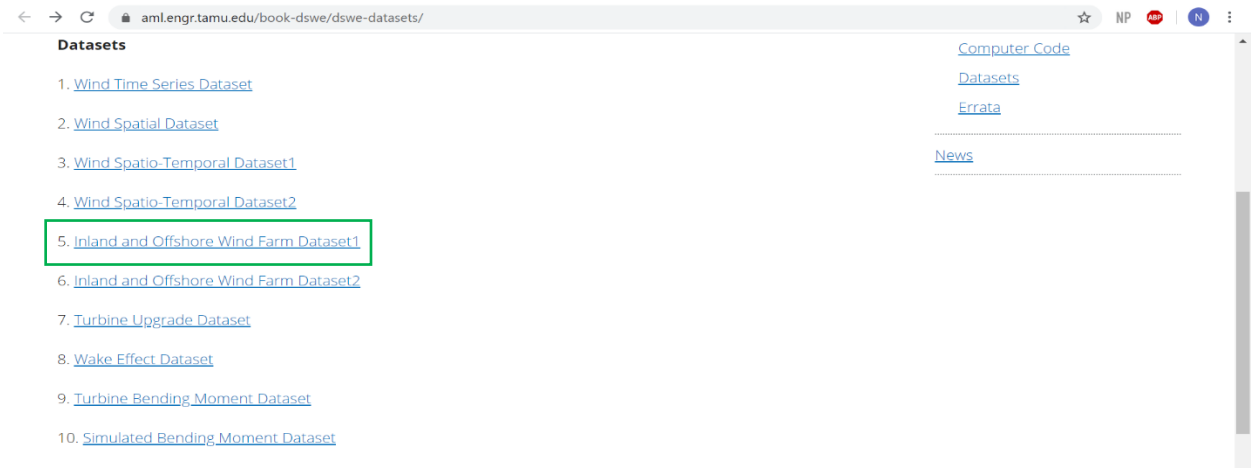
In the table, the results of AMK, KNN, BART, SSANOVA, and the binning method are taken directly from Table 5.7 of the *Data Science for Wind Energy* book, the results of tempGP, SVM and DNN are obtained using the respective Python package functions.

	Temp GP	AMK	KNN	BART	DNN	SSANOVA	SVM	BIN
WT1	0.064	0.074	0.077	0.076	0.082	0.087	0.096	0.131
WT2	0.070	0.080	0.083	0.082	0.088	0.091	0.091	0.116
WT3	0.055	0.065	0.067	0.067	0.074	0.077	0.085	0.122
WT4	0.079	0.100	0.104	0.104	0.111	0.112	0.117	0.152
WT5	0.065	0.079	0.081	0.088	0.089	0.095	0.088	0.097
WT6	0.066	0.080	0.083	0.091	0.094	0.104	0.094	0.109
Average	0.066	0.080	0.082	0.084	0.090	0.094	0.095	0.121
Relative to TempGP	-	1.20	1.24	1.27	1.35	1.42	1.44	1.82

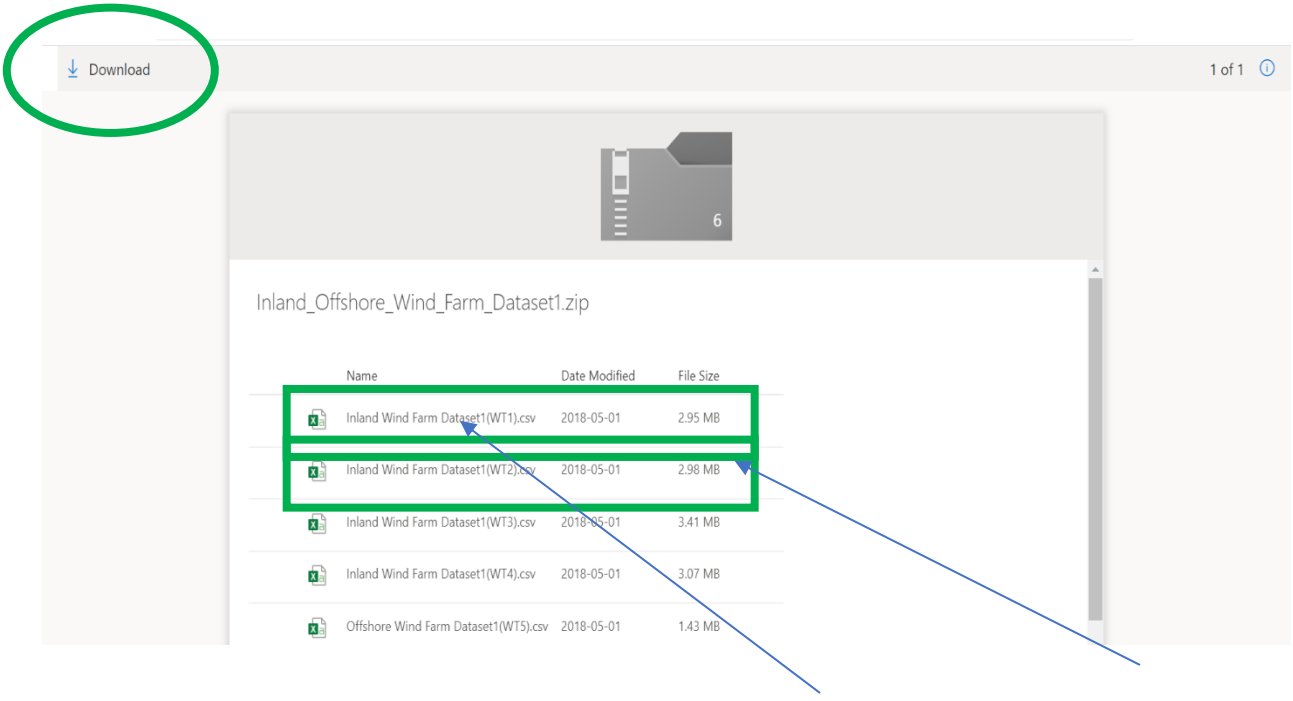
10 How to select the subsets of data, before and after a decision point, so that they can be deemed probabilistically comparable?

Step1: Download the sample data set.

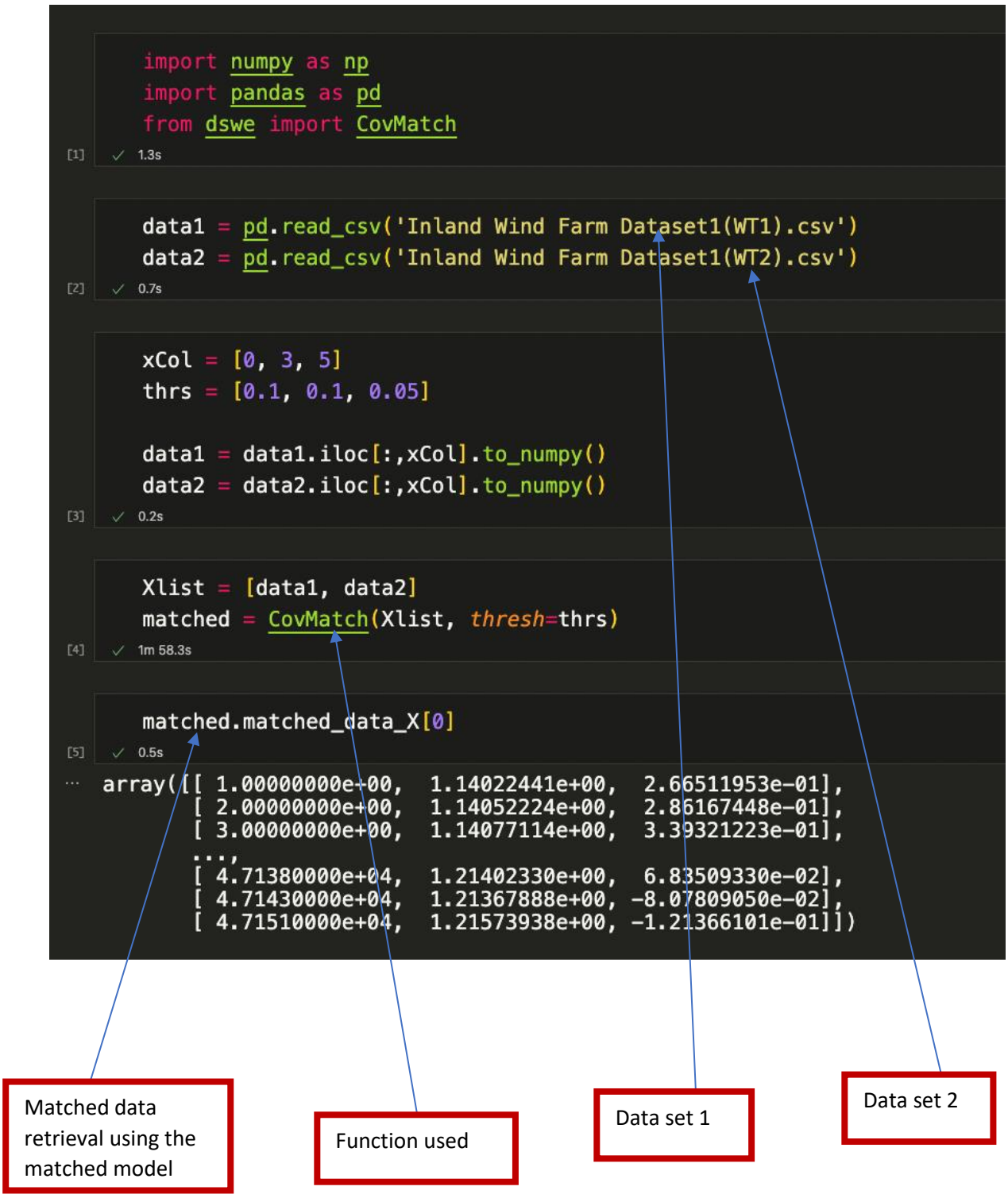
Visit site using following link - *https://aml.engr.tamu.edu/book-dswe/dswe-datasets/*. The page looks like as shown below and select the option 5.



Download the data set as shown below in green boxes. After downloading, save the file in working directory.



Step 3: The package is loaded and data is imported. Also, arguments are prepared and matching function is employed as shown



11 How to compare performance of two turbine or two data set in different time period?

Step 1: The data set mentioned in previous question will be used to demonstrate the performance quantification. The package is loaded and data is imported as shown.

```
import numpy as np
import pandas as pd
from dswe.comparePCurve import ComparePCurve
```
[1] ✓ 1.2s

```
data1 = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
data2 = pd.read_csv('Inland Wind Farm Dataset1(WT2).csv')
```
[2] ✓ 0.7s

Step 2: Prepare the arguments to use performance comparison function as shown.

```
xCol = [0, 3]
thresh = [0.1, 0.1]
testcol = [0, 1]
circ_pos = [1]

Xlist = [data1.iloc[:1000,xCol].to_numpy(), data2.iloc[:1000,xCol].to_numpy()]
ylist = [data1.iloc[:1000,6].to_numpy(), data1.iloc[:1000,6].to_numpy()]
```
[3] ✓ 0.3s

```
model = ComparePCurve(Xlist, ylist, testcol, circ_pos=circ_pos, thresh=thresh)
```
[ ]

```
model.weighted_diff, model.scaled_diff
```
[6] ✓ 0.2s

```
(8.13, 39.16)
```

Result retrieval

Function used

Argument preparation

Data set

How to use a different probability distribution than that computed from the data to compute the weighted difference between the power curves?

Step 1: Construct a desired ***testset*** and a probability distribution over that ***testset*** as shown below as an example:

```
ws_min = 5 # minimum value of wind speed for constructing the testset.
ws_max = 15 # maximum value of wind speed for constructing the testset.
ws_test = np.linspace(ws_min, ws_max, 50) # generate 50 grid points for wind speed.
rho_min = 1.1 # minimum value of air density for constructing the testset.
rho_max = 1.3 # minimum value of air density for constructing the testset.
rho_test = np.linspace(rho_min, rho_max, 50) # generate 50 grid points for air density.
testset = np.array([[m, n] for n in rho_test for m in ws_test])

# For example we use a Weibull distribution for wind speed with shape = 2.25 and scale = 6.5.
# and uniform distribution for air density. Please change as desired.
# Multiply the weights by 1 to denote a uniform distribution for air density.
from scipy.stats import dweibull
weights = dweibull.pdf(testset[:,0], c=2.25, scale=6.5)
weights = weights/sum(weights) # normalizing weights to ensure that they sum to 1.
✓ 0.4s
```

Step 2: Run ***ComparePCurve()*** function as described earlier with the ***testset*** generated in Step 1 as one of the inputs:

```
data1 = pd.read_csv('Inland Wind Farm Dataset1(WT1).csv')
data2 = pd.read_csv('Inland Wind Farm Dataset1(WT2).csv')

Xlist = [data1.iloc[:1000, [0,3,5]].to_numpy(), data2.iloc[:1000, [0,3,5]].to_numpy()]
ylist = [data1.iloc[:1000, 6].to_numpy(), data1.iloc[:1000, 6].to_numpy()]

model = ComparePCurve(Xlist, ylist, [0,1], circ_pos=1)
```

Step 3: Use the output from ***ComparePCurve()*** function to compute the weighted difference and statistically significant weighted difference with the weights computed in Step 1 as follows:

```
# Computing weighted difference with mu1 as the base for percentage calculation
weighted_diff = model.compute_weighted_difference(weights=weights, baseline=1)
conf_band = model.band

# Computing statistically significant weighted difference
weighted_stat_diff = model.compute_weighted_difference(weights=weights, stat_diff=True)
✓ 0.5s
```

12 A case study of estimating the effect associated with turbine upgrades.

This case study applies the functions in the DSWE package to the Turbine Upgrade Dataset, associated with the book, *Data Science for Wind Energy*, and available from the website below. The case study is explained in Section 4.1 of the preprint https://arxiv.org/pdf/2005.08652.pdf. Additional information about the dataset and turbine upgrades can be found in Section 1.2.3 and Chapter 7 of *Data Science for Wind Energy.* The dataset include two cases of upgrades–one is the pitch angle adjustment and the second is the vortex generator installation. The steps below explain how the top rows of Table 3 of the preprint https://arxiv.org/pdf/2005.08652.pdf are obtained as well as how the VG effect is estimated.

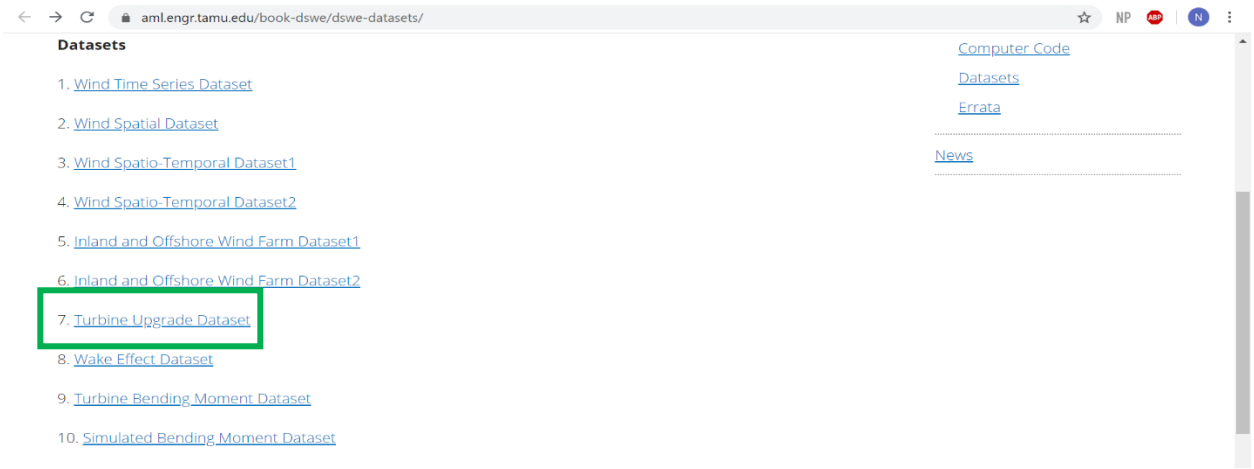
The above preprint is now published in the journal of *Renewable Energy*. The paper's full citation is

> Ding, Kumar, Prakash, Kio, Liu, Liu, and Li, 2021, "A case study of space-time performance comparison of wind turbines on a wind farm," *Renewable Energy*, Vol. 171, pp. 735-746.

One can go to https://aml.engr.tamu.edu/2001/09/01/publications/ (and then go to J77) to get the reproducibility report and R code for reproducing the majority of the results in this paper.

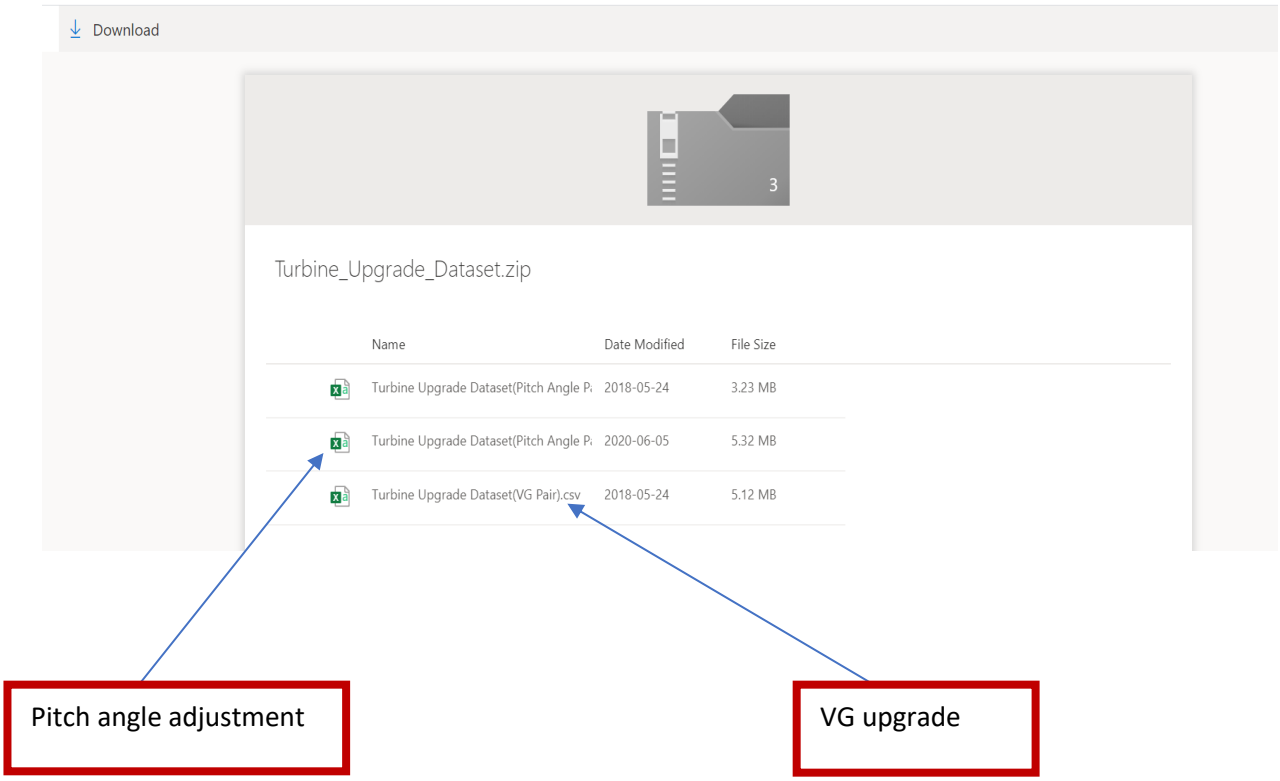
Step1: Download the sample data set as shown.

Visit site using the following link - *https://aml.engr.tamu.edu/book-dswe/dswe-datasets/*. The page looks like as shown below and select the option 7.



Download the sample data set as shown below. After downloading, save the file in working

directory.



Step 2: Set the path containing data set to a current working directory. Further load the package and import the data set as shown

For pitch angle pair:

```
[1]  import dswe
     import pandas as pd
     ✓ 1.2s

[2]  df = pd.read_csv('Turbine_Upgrade_Dataset/Turbine Upgrade Dataset(Pitch Angle Pair, Table7.3).csv')
     ✓ 0.8s
```

For VG upgrade:

```
[1]  import dswe
     import pandas as pd
     ✓ 1.1s

[2]  df = pd.read_csv('Turbine_Upgrade_Dataset/Turbine Upgrade Dataset(VG Pair).csv')
     ✓ 0.8s
```

Step 3: Use the performance comparison function on pitch angle adjustment and VG upgrade as shown below. In case of pitch angle adjustment, user just needs to import the appropriate data set and change the value of 'input' variable as shown below in the script

For Pitch Angle adjustment

```
import pandas as pd
import numpy as np
from dswe import ComparePCurve
```

1.1s

```
data = pd.read_csv('Turbine_Upgrade_Dataset/Turbine Upgrade Dataset(Pitch Angle Pair, Table7.3).csv')
data1 = data.loc[data['upgrade.status'] == 0, :]
data2 = data.loc[data['upgrade.status'] == 1, :]

r = range(2, 10)
r_prime = [1.25, 1.87, 2.49, 3.11, 3.74, 4.36, 4.98, 5.60]

x_col = [2, 3, 4, 5, 6]
y_col_control = 17
x_col_circ = [1]
test_col = [0, 1]

Xlist = [data1.iloc[:, x_col], data2.iloc[:, x_col]]
ylist_control = [data1.iloc[:, [y_col_control]],
                 data2.iloc[:, [y_col_control]]]

control = ComparePCurve(Xlist, ylist_control, test_col, circ_pos=x_col_circ, thresh=0.2, grid_size=[50, 50])
```

Control model

```
results = pd.DataFrame(index=['r_prime', 'Our estimate', 'Our estimate/r_prime'],
                       columns=[str(i) + '%' for i in r], dtype=np.float16)
results.iloc[0, :] = r_prime

for i, y_col_test in enumerate(range(9, 17)):
    ylist_test = [data1.iloc[:, [y_col_test]], data2.iloc[:, [y_col_test]]]
    test = ComparePCurve(Xlist, ylist_test, test_col, circ_pos=x_col_circ, thresh=0.2, grid_size=[50, 50])
    results.iloc[1, i] = test.weighted_diff - control.weighted_diff
    results.iloc[2, i] = results.iloc[1, i]/results.iloc[0, i]
```

```
results
```

0.4s

	2%	3%	4%	5%	6%	7%	8%	9%
r_prime	1.250000	1.870117	2.490234	3.109375	3.740234	4.359375	4.980469	5.601562
Our estimate	1.429688	2.070312	2.720703	3.369141	4.019531	4.671875	5.308594	6.140625
Our estimate/r_prime	1.143555	1.107422	1.092773	1.083984	1.074219	1.071289	1.065430	1.096680

Result display

Test model

For VG upgrade:

Result display

Target for control and test model